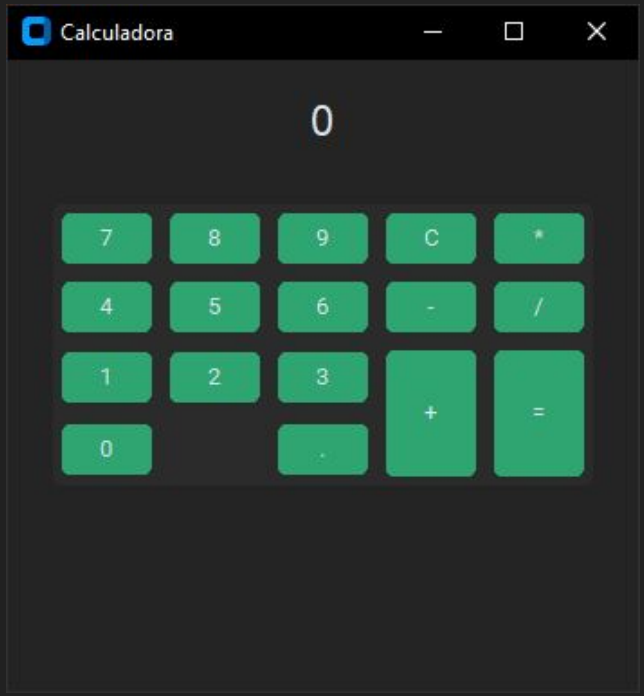
Tipos de testes

Testes de caixa preta

Exemplo:



Testes de caixa preta

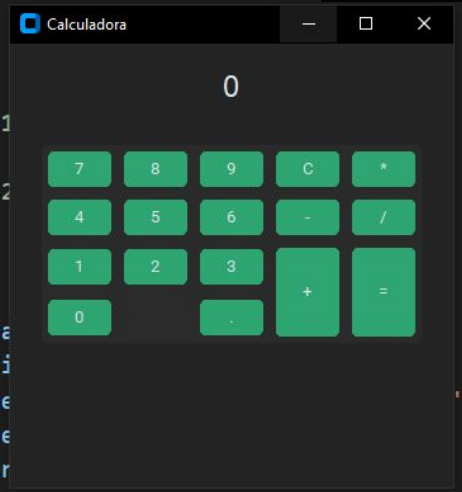
Caracteristicas:

- Não temos acesso ao código
- Complexidade baixa
- Executa-se a aplicação e obtém o retorno
- Mais trabalhoso para localizar os erros

Testes de caixa branca

Exemplo:

```
59
60  def _calcular(expression):
61      operacoes = ['*', '/', '+', '-']
62      try:
63          for operacao in operacoes:
64              if expression.count(operacao) > 1
65                  raise
66              if expression.count(operacao) > 2
67                  raise
68          for operacao in operacoes:
69              if operacao in expression:
70                  if len(expression.split(opera
71                      valor1, valor2 = expressi
72                  elif len(expression.split(ope                    ':
73                      _, valor1, valor2 = expre
74                      valor1 = operacao + valor
75                  else:
76                      raise
77                  return identificar_operacao(valor1, operacao, valor2)
78      except Exception:
79          return 'Erro'
80
```



Testes de caixa branca

Caracteristicas:

- Temos acesso ao código ou ao banco de dados
- Executa-se a aplicação e obtém o retorno
- Mais trabalhoso para localizar os erros
- Pouco complexo

Testes de unidade (ou unitário)

exemplo:

```python
@pytest.mark.parametrize(
    "email,resultado_esperado",
    [
        ("email@email.com", None),
        ("email@teste.edu.br", None),
        ("emailcommaisde50caracteresemailcommaisde50caracteres", "o email pode ter somente 50 caracteres"),
        ("emailsemarroba.com.br", "email precisa ter @"),
        ("emailcom@massemponto", "email precisa ter . após o @"),
        ("emailcom@ecom.terminadoem.", "email não pode terminar em ."),
        ("email@.com", "não pode ter . logo após o @")
    ]
)
def test_email(cadastro, email, resultado_esperado):
    assert cadastro._valida_email(email=email) == resultado_esperado


def test_email_2(cadastro):
    assert cadastro._valida_email(email="email@email.com") == None
    assert cadastro._valida_email(email="email@teste.edu.br") == None
    assert cadastro._valida_email(
        email="emailcommaisde50caracteresemailcommaisde50caracteres") == "o email pode ter somente 50 caracteres"
    assert cadastro._valida_email(email="emailsemarroba.com.br") == "email precisa ter @"
    assert cadastro._valida_email(email="emailcom@massemponto") == "email precisa ter . após o @"
    assert cadastro._valida_email(email="emailcom@ecom.terminadoem.") == "email não pode terminar em ."
    assert cadastro._valida_email(email="email@.com") == "não pode ter . logo após o @"
```

Testes de unidade (ou unitário)

Caracteristicas:

- Rapido para executar
- Complexidade baixa
- Não há dependências
- Geralmente é feito em grande quantidade
- Geralmente testa todas as validações do método

Testes de componente

Exemplo:

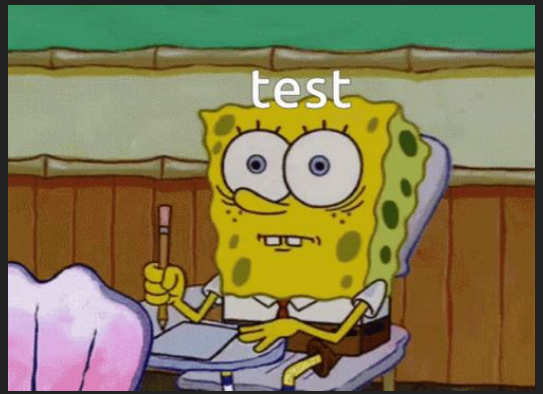
```python
# testes de componente
@pytest.mark.parametrize(
    "pessoa,conf_senha,resultado_esperado",
    [
        (Pessoa(nome="nome teste", email="email@email.com", cpf="76739421005", senha="Teste@1234"), "Teste@1234", None),
        (Pessoa(nome="", email="email", cpf="123", senha="senha"), "", "Preencha todos os campos"),
        (Pessoa(nome="nome", email="email", cpf="123", senha="senha"), "", "preencha o sobrenome também"),
        (Pessoa(nome="nome teste", email="email", cpf="123", senha="senha"), "", "email precisa ter @"),
        (Pessoa(nome="nome teste", email="email@email.com", cpf="123", senha="senha"), "", "CPF inválido"),
        (
            Pessoa(nome="nome teste", email="email@email.com", cpf="76739421005", senha="Teste"),
            "",
            "é necessário possuir caracter especial"
        ),
        (
            Pessoa(nome="nome teste", email="email@email.com", cpf="76739421005", senha="Teste@1234"),
            "",
            "As senhas não coincidem!"
        ),

    ]
)
def test_validacao(cadastro, pessoa, conf_senha, resultado_esperado):
    assert cadastro.validacao(pessoa=pessoa, confirma_senha=conf_senha) == resultado_esperado
```

Testes de componente

Caracteristicas:

- Rapido para executar
- Complexidade baixa à intermediária
- Verifica interação entre os métodos e classes
- A quantidade varia dependendo da complexidade
- Geralmente testa poucos casos do método



Teste de integração

Exemplo:

```python
@pytest.mark.parametrize(
    "pessoa_alt,resultado_esperado",
    [
        (
            Pessoa(id=1, nome="novo nome", email="", cpf="", senha="Teste!23"),
            (True, "Atualizado com sucesso!"),
        ),
        (
            Pessoa(id=1, nome="novo nome", email="", cpf="", senha="NovaSenha!23"),
            (True, "Atualizado com sucesso!"),
        ),
        (
            Pessoa(id=1, nome="nome_errado", email="", cpf="", senha="Teste!23"),
            (False, "preencha o sobrenome também"),
        ),
        (
            Pessoa(id=1, nome="novo nome", email="", cpf="", senha="senhasemnumeros"),
            (False, "é necessário possuir caracter especial"),
        ),
    ]
)
def test_alterar_cadastro(cadastro, insere_cadastro, pessoa_alt, resultado_esperado):
    resultado = cadastro.atualizar_cadastro(id=pessoa_alt.id, nome=pessoa_alt.nome, senha=pessoa_alt.senha)
    assert resultado == resultado_esperado

```

Teste de integração

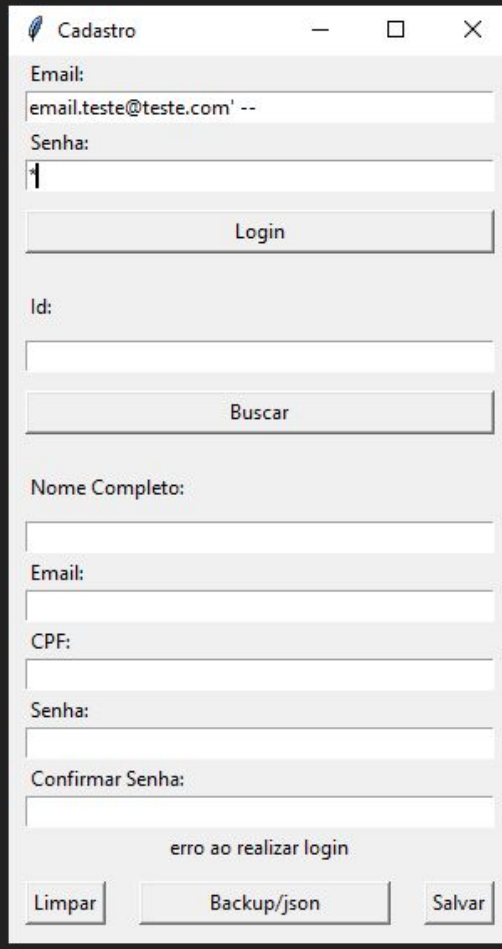
Caracteristicas:

- Teste pode ser mais lento
- Complexidade intermediária à alta
- Verifica interação entre os processos (banco de dados, fila, etc)
- Poucos testes que fazem ações mais complexas

Testes de segurança

Exemplo:

- Injeção de sql



Testes de segurança

Caracteristicas:

- Tenta explorar falhas de segurança
- Complexidade alta
- Tenta obter acessos não fornecidos

Testes de desempenho

Exemplo:

```python
import os
from datetime import datetime
from cadastro import Cadastro, Pessoa
from sqlite_banco import Banco


os.remove('teste_desenpenho.db')
cadastro = Cadastro(banco=Banco(nome_banco='teste_desenpenho.db'))

quantidade = 10000

inicio = datetime.now()
senha = "Senha@123"
pessoa = Pessoa(nome="nome teste", email="", cpf="", senha=senha)

for indice in range(0, quantidade):
    pessoa.email = f"email{indice}@teste.com"
    pessoa.cpf = str(indice + 1).zfill(11)
    sucesso, mensagem = cadastro.cadastrar(pessoa=pessoa, confirma_senha=senha)
    assert sucesso
final = datetime.now()
print(f"[INSERSÃO] Tempo para inserir {quantidade} registros = {final - inicio}")
inicio = datetime.now()
todos_cadastros = cadastro.banco.buscar_todos_os_cadastros()
assert len(todos_cadastros) == quantidade
final = datetime.now()
print(f"[CONSULTA] Tempo para consultart {quantidade} registros = {final - inicio}")
os.remove('teste_desenpenho.db')
```

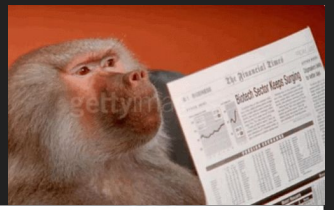
```
[INSERSÃO] Tempo para inserir 10000 registros = 0:08:35.331629
SELECT * FROM usuario
[CONSULTA] Tempo para consultart 10000 registros = 0:00:00.113309
```

Testes de desempenho

Caracteristicas:

- Visa realizar repetições para identificar o tempo de processo
- Complexidade media
- Não é necessário testar todas as funcionalidades

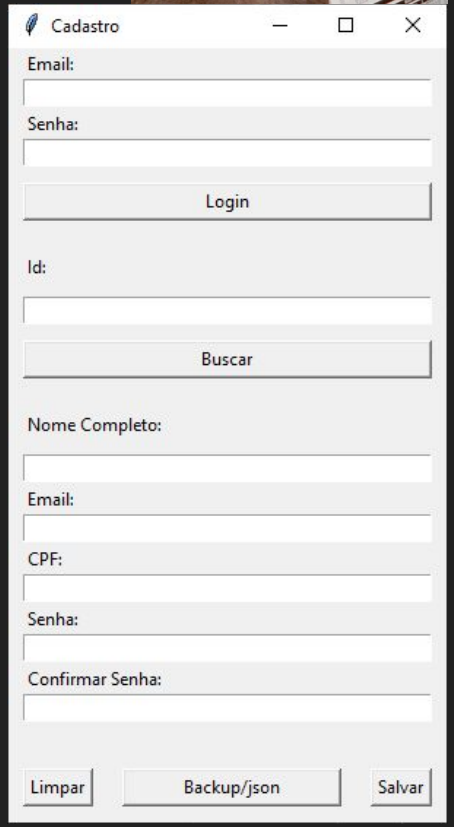
Testes de aceitação



Exemplo:

Requisito de aceitação:

- Deve permitir cadastrar um usuario com:
  - Nome, email, cpf e senha
- Ao cadastrar deve retornar o identificador
- Deve ser possível buscar o cadastro por id
- Deve permitir alterar o nome e a senha
- Deve permitir realizar o login com email e senha
- Deve permitir realizar o backup dos dados cadastrados



Testes de aceitação

Caracteristicas:

- Usuário informa as necessidades
- Complexidade baixa a media
- Geralmente realizada com solicitante